UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/681,759	10/08/2003	Robert Anthony DeLine	MS304376.1	8309

27195 7590 01/23/2007
AMIN. TUROCY & CALVIN, LLP
24TH FLOOR, NATIONAL CITY CENTER
1900 EAST NINTH STREET
CLEVELAND, OH 44114

EXAMINER	
PHAM, THAI V	
ART UNIT	PAPER NUMBER
2192	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	01/23/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)
	10/681,759	DELINE ET AL.
	Examiner	Art Unit
	Thai Van Pham	2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *08 October 2003*.

2a)□ This action is **FINAL**. 2b)☒ This action is non-final.

3)□ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)□ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-24* is/are rejected.

7)☒ Claim(s) *19,23* is/are objected to.

8)□ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)□ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *08 October 2003* is/are: a)□ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)□ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)□ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)□ All b)□ Some * c)□ None of:

1.□ Certified copies of the priority documents have been received.

2.□ Certified copies of the priority documents have been received in Application No. _____.

3.□ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) □ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO/SB/08)
Paper No(s)/Mail Date *04/05/04, 03/20/06*.

4) □ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____ .

5) □ Notice of Informal Patent Application

6) □ Other: _____.

DETAILED ACTION

This is the initial office action based on the application filed on 10/08/2003.

Priority date that has been considered for this application is 10/08/2003.

Claims 1 – 24 are currently pending and have been considered below.

Drawing

1. The drawing is objected to because of the following informalities: typographical errors.

-- FIG. 16, item 1630: "...THE [CUSTOME] CUSTOM STATE SUBCLASS."

-- FIG. 17, item 1730: "...EXECUTABLE [CIDE] CODE."

Appropriate correction is required.

Claim Objections

2. Claims 19 and 23 are objected to because of the following informalities: typographical error(s). The typo is bracketed and the assumed proper correction is underlined as below.

-- Claim 19.

In the third limitation of the claim: "*...determining whether a fault condition exists based, at least in part, upon the information from the [per] pre-condition plug-in ...* ". Appropriate correction is required.

-- Claim 23.

In the first limitation of the claim: "*means for [that] receiving [and] a specification* ... ".

Claim Rejections - 35 USC § 101

35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

3. Claims 1 – 14, 21, and 23 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

-- Claims 1 and 23.

The claims recite: "An executable code check system..." as the claimed subject matter. The claimed system is directed to a software system where the means for performing the functionality of the system's components are made up of software instructions and modules. Thus, the claimed system is considered a software program, per se. See MPEP 2106.01 (I): "Descriptive material can be characterized as either "functional descriptive material" or "nonfunctional descriptive material." In this context, "functional descriptive material" consists of data structures and computer programs which impart functionality when employed as a computer component. (The definition of "data structure" is "a physical or logical relationship among data elements, designed to support specific data manipulation functions." The New IEEE Standard Dictionary of Electrical and Electronics Terms 308 (5th ed. 1993).)) ... ".

-- Claims 2 – 14.

Claims 2 – 14 all fail to remedy the nonstatutory claimed subject matter of claim 1, and therefore, are also nonstatutory.

-- <u>Claim 21</u>.

Claim 21 is directed to a data packet, i.e., data structure, as the claimed subject matter. Data structures not claimed as embodied in computer-readable media are descriptive material per se and are not statutory because they are not capable of causing functional change in the computer (See MPEP 2106.01 – I).

Claim Rejections - 35 USC § 112

The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

4. Claim 23 is rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

-- <u>Claim 23</u>.

The claim recites the limitation "*the object file*" in the first limitation of the claim. There is insufficient antecedent basis for this limitation in the claim. In the principle of compact prosecution, Examiner anticipates the claim to be corrected as: "...*a specification associated with [the]* <u>*an*</u> *object file...* ".

Double Patenting

5. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. A nonstatutory obviousness-type double patenting rejection is appropriate where the conflicting claims are not identical, but at least one examined application claim is not patentably distinct from the reference claim(s) because the examined application claim is either anticipated by, or would have been obvious over, the reference claim(s). See, e.g., *In re Berg*, 140 F.3d 1428, 46 USPQ2d 1226 (Fed. Cir. 1998); *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) or 1.321(d) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent either is shown to be commonly owned with this application, or claims an invention made as a result of activities undertaken within the scope of a joint research agreement.

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

6. Claims 1, 15, 21, 22, and 23 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claims 5 + 6, 20, 22, 24, and 25 of copending Application No. 10/667,542.

This is a provisional obviousness-type double patenting rejection because the conflicting claims have not in fact been patented.

Although the conflicting claims are not identical, they are not patentably distinct from each other for the following reasons.

- Claims 1, 21, 22, and 23 of the instant application recite: "a specification associated with the object file", whereas claims 5, 22, 24, and 25 of the copending application recite: "executable code having an embedded specification". A specification that is **embedded** in an executable code, i.e. **an object file**, is obviously **associated** with the executable code or **the object file**.

- Claims 1, 15, 21, 22 and 23 of the instant application further describe: "the specification comprising information associated with **a plug-in condition for a method**", whereas claims 5 and 6 of the copending application further describe the embedded specification contains information related to **a finite state machine associated with a method at a pre-state and a post-state as well as the transition between them**. The finite state machine information here is obviously one type of plug-in condition, i.e. pre/post condition(s), recited in the instant application. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to broadly state the information contained in the specification on a finite state machine and associated method(s) as plug-in conditions for the method(s).

Instant Application 10/681,759	**Copending Application** 10/667,542
Claim 1. An executable code check system comprising: • an input component that receives an object file and a specification ***associated*** with the object file, ***the specification comprising information associated with a plug-in condition for a method***; and • a checker that employs the specification to facilitate static checking of the object file, the checker providing information if a fault condition is determined.	Claim 5. The system of claim 4, wherein: • the method order is constrained by specifying a finite state machine in which the states have symbolic names and transitions between state are labeled with method names. Claim 6: The system of claim 1, the specification comprising • a state machine protocol wherein a method specifies a pre-state and a post-state.
Claim 15. A method of facilitating static checking of executable code comprising: • receiving executable code; • receiving a specification associated with the executable code, ***the specification comprising information associated with at least one of a precondition and a postcondition for a method***; • statically applying the specification to the executable code; • determining whether a fault condition exists based, at least in part, upon the statically applied specification; and, providing information associated with the fault condition, if a fault condition is determined to exist.	Claim 20. A method of facilitating static checking of executable code comprising: • receiving executable code; • retrieving a specification associated with the executable code; • statically applying the specification to the executable code; • determining whether a fault condition exists based, at least in part, upon the statically applied specification; and, providing information associated with the fault condition, if a fault condition is determined to exist.
Claim 21. A data packet transmitted between two or more computer components that facilitates static checking of executable code, the data packet comprising: • a specification ***associated*** with executable code, ***the specification comprising information associated with at least one of a plug-in precondition and a plug-in postcondition***,	Claim 22. A data packet transmitted between two or more computer components that facilitates static checking of executable code, the data packet comprising: • executable code having an ***embedded*** specification,

• the specification providing information to be employed to statically check the executable code.	• the embedded specification providing information to be employed to statically check the executable code.
Claim 22. A computer readable medium storing computer executable components of an executable code check system comprising: • an input component that receives an object file and a specification ***associated*** with the object file, ***the specification comprising information associated with a plug-in condition for a method***; and • a checker component that employs the specification to facilitate static checking of the object file, the checker component providing information if a fault condition is determined.	Claim 24. A computer readable medium storing computer executable components of an executable code check system comprising: • an input component that receives an object file having an ***embedded*** specification; and • a checker component that employs the specification to facilitate static checking of the object file, the checker providing information if a fault condition is determined.
Claim 23. An executable code check system comprising: • means for that receiving and a specification ***associated*** with an object file, ***the specification comprising information associated with a plug-in condition for a method***; • means for statically checking the object file based, at least in part, upon the specification and determining if a fault condition exists; and • means for providing information if a fault condition is determined to exist.	Claim 25. An executable code check system comprising: • means for receiving an object file having an ***embedded*** specification; • means for statically checking the object file based, at least in part, upon the embedded specification and determining if a fault condition exists; and • means for providing information if a fault condition is determined to exist.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

7. Claims 1 – 6, 12, 13, 15, 16, and 19 – 24 are rejected under 35 U.S.C. 102(b) as being anticipated by **Necula et al**. (US 6,128,774).

-- Claim 1.

Necula discloses *an executable code check system comprising*:

- *an input component that receives an object file and a specification associated with the object file*;

(Figs. 3 and 4 – VCGen module **32** – and associated text, e.g., Col. 4: lines 36 – 38; "The VCGen module **32** performs two tasks. First, it checks simple safety properties of the annotated executable **30** ...Second, the VCGen module **32** watches for instructions whose execution might violate the safety policy."
Examiner notes that the object file is synonymous to the executable file, and the specification associated with the object file is the annotation embedded in the executable and/or rules defined by the safety policy.)

- *the specification comprising information associated with a plug-in condition for a method*; and

(Fig. 4 – configuration data **50** – and associated text , e.g., Col. 7: lines 19 – 32; "...The configuration data **50** also describes, by precondition-postcondition pairs, all of the functions that the untrusted code **42** are permitted to invoke.")

- *a checker that employs the specification to facilitate static checking of the object file, the checker providing information if a fault condition is determined.*

(Fig. 4 – VCGen 32 – and associated text, e.g., Col. 13: lines 16 – 25; "The safety predicate **34** for a function is obtained by evaluating it symbolically starting in a state that maps the global variables ...and function formal parameters to new variables ... ". Figs. 3 and 5 – proof checker **40** – and associated text, e.g., Col. 13: lines 25 – 35; "FIG. 5 is a diagram illustrating an implementation of the proof checker module **40** of FIG. 3. The function of the proof checker module **40** is to verify that the proof **38** supplied by the untrusted proof producer uses only allowed axioms and inference rules and that it is a proof of the required safety predicate **34** ... ". VCGen and proof checker performs the checking of the executable based on the provided specification is performed statically.)

-- Claim 2.

Necula discloses *the system of claim 1,*

- *the plug-in condition comprising a precondition for the method.*

(Fig. 4 – configuration data **50** – and associated text , e.g., Col. 7: lines 19 – 32; "...The configuration data **50** also describes, by precondition-postcondition pairs, all of the functions that the untrusted code **42** are permitted to invoke.")

-- Claim 3.

Necula discloses *the system of claim 2,*

- *the checker providing information associated with an object's state after a call to the method, the information being based, at least in part, upon the plug-in precondition.*

(Fig. 4 – VCGen 32 – and associated text, e.g., Col. 13: lines 16 – 25; "The safety predicate **34** for a function is obtained by evaluating it symbolically starting in a state that maps the global variables ...and function formal parameters to new variables ... ")

-- Claim 4.

Necula discloses *the system of claim 1,*

- *the plug-in condition comprising a postcondition for the method.*

(Fig. 4 – configuration data **50** – and associated text , e.g., Col. 7: lines 19 – 32; "...The configuration data **50** also describes, by precondition-postcondition pairs, all of the functions that the untrusted code **42** are permitted to invoke.")

-- Claim 5.

Necula discloses *the system of claim 4*,

- *the checker providing information associated with an object's state after a call to the method, the information being based, at least in part, upon the plug-in postcondition.*

(Fig. 4 – VCGen 32 – and associated text, e.g., Col. 13: lines 16 – 25; "The safety predicate **34** for a function is obtained by evaluating it symbolically starting in a state that maps the global variables ...and function formal parameters to new variables ... ")

-- Claim 6.

Necula discloses *the system of claim 1,*

- *the object file being based, at least in part, upon a language that compiles to Common Language Runtime.*

(Col. 1: lines 40 – 55; "High level type-safe programming languages, such as ML and Java ... in practice programs often have some components written in ML or Java and other components written in different languages (e.g. C or assembly language)." According to **Meijer et al**. "Technical Overview of the Common Language Runtime", CLR, which is expressed in the Common Intermediate Language (CIL), can be compiled from a language such as C, Pascal, etc.)

-- Claim 12.

Necula discloses *the system of claim 1, wherein*

- *the specification is embedded with the object file.*

(Fig. 4 – annotated executable **30** – and associated text, e.g., Col. 6. The annotations are embedded with the executable.)

-- Claim 13.

Necula discloses *the system of claim 1, wherein*

- *the specification is stored in a specification repository.*

(Fig. 4 – configuration data **50** – and associated text , e.g., Col. 6: lines 57 – 59; "...a file of configuration data **50**, which is provided as part of the safety policy by the code consumer." A file is stored in memory, e.g. a file repository, and separate from the executable.)

-- Claim 15.

Necula discloses *a method of facilitating static checking of executable code comprising*:

- receiving executable code;

(Fig. 3 – annotated executable **30** – and associated text, e.g., Col. 4: lines 32 – 35. Examiner notes that the object file is synonymous to the executable file.)

- receiving a specification associated with the executable code; the specification comprising information associated with at least one of a precondition and a postcondition for a method;

(Figs. 3 and 4 – VCGen module **32** – and associated text, e.g., Col. 4: lines 36 – 38; "The VCGen module **32** performs two tasks. First, it checks simple safety properties of the annotated executable **30** ...Second, the VCGen module **32** watches for instructions whose execution might violate the safety policy."

Fig. 4 – configuration data **50** – and associated text , e.g., Col. 7: lines 19 – 32; "...The configuration data **50** also describes, by precondition-postcondition pairs, all of the functions that the untrusted code **42** are permitted to invoke."

Examiner notes that the specification associated with the object file is the annotation embedded in the executable and/or rules defined by the safety policy.)

- statically applying the specification to the executable code;

(Fig. 4 – VCGen 32 – and associated text, e.g., Col. 13: lines 16 – 25; "The safety predicate **34** for a function is obtained by evaluating it symbolically starting in a state that maps the global variables ...and function formal parameters to new variables ... ".

Figs. 3 and 5 – proof checker **40** – and associated text, e.g., Col. 13: lines 25 – 35; "FIG. 5 is a diagram illustrating an implementation of the proof checker module **40** of FIG. 3. The function of the proof checker module **40** is to verify that the proof **38** supplied by the untrusted proof producer uses only allowed axioms and inference rules and that it is a proof of the required safety predicate **34** ... ". VCGen and proof checker performs the checking of the executable based on the provided specification is performed statically.)

• determining whether a fault condition exists based, at least in part, upon the statically applied specification; and, providing information associated with the fault condition, if a fault condition is determined to exist.

(Fig. 4 – Safety Predicate (SP) – and associate text, e.g. Col. 4: lines 40 – 48; "the VCGen module **32** emits a predicate that expresses the conditions under which the execution of the instruction is safe ... ".)

-- Claim 16.

Necula discloses *a computer readable medium having stored thereon computer executable instructions for carrying out the method of claim 15.*

(Fig. 6, Col 16: lines 25 – 30; "Software verification modules **66**, of the type disclosed herein in conjunction with the present invention, are stored in the computer storage devices **64**... ".)

-- Claim 19.

Necula discloses *a method of performing static checking of executable code comprising*:

• invoking a precondition plug-in, providing the precondition plug-in with a program execution state; and receiving information from the precondition plug-in;

(Col. 5: lines 50 – 60; "...the code consumer can declare a precondition, which is essentially a description of the calling convention the consumer will use when invoking the untrusted code...")

• determining whether a fault condition exists based, at least in part, upon the information from the pre-condition plug-in; and

(Col. 7: lines 21 – 32; "...the code consumer guarantees that the precondition holds when the untrusted code **42** is invoked ...The precondition for such a function is a predicate that the untrusted code **42** must establish before calling the function ... ".)

• providing information associated with the fault condition, if a fault condition is determined to exist.

(Fig. 4 – Safety Predicate (SP) – and associate text, e.g. Col. 4: lines 40 – 48; "the VCGen module **32** emits a predicate that expresses the conditions under which the execution of the instruction is safe ... ".)

-- Claim 20.

Necula discloses *the method of claim 19, further comprising at least one of the following*:

• *invoking a postcondition plug-in, providing the postcondition plug-in with the program execution state; and receiving information from the postcondition plug-in.*

(Col. 5: lines 50 – 60; "...postconditions for the untrusted code. These are constraints on the final execution state of the untrusted code...

Col. 7: lines 21 – 32; "The untrusted code **42** must ensure that the postcondition holds on return ...the postcondition is predicate that the untrusted code **42** may assume to hold upon return from the function.")

-- Claim 21.

Necula discloses a data packet (Fig. 4 – configuration data file **50**) transmitted between two or more computer components (Fig. 6 – workstations **54** connected to a communication channel **56**) that facilitates static checking of executable code, the data packet comprising:

- a specification associated with executable code, the specification comprising information associated with at least one of a plug-in precondition and a plug-in postcondition, the specification providing information to be employed to statically check the executable code.

(Figs. 3 and 4 – VCGen module **32** – and associated text, e.g., Col. 4: lines 36 – 38; "The VCGen module **32** performs two tasks. First, it checks simple safety properties of the annotated executable **30** ...Second, the VCGen module **32** watches for instructions whose execution might violate the safety policy."

Examiner notes that the object file is synonymous to the executable file, and the specification associated with the object file is the annotation embedded in the executable and/or rules defined by the safety policy.

Fig. 4 – configuration data **50** – and associated text , e.g., Col. 7: lines 19 – 32; "...The configuration data **50** also describes, by precondition-postcondition pairs, all of the functions that the untrusted code **42** are permitted to invoke."

Fig. 4 – VCGen 32 – and associated text, e.g., Col. 13: lines 16 – 25; "The safety predicate **34** for a function is obtained by evaluating it symbolically starting in a state that maps the global variables ...and function formal parameters to new variables ... ". Figs. 3 and 5 – proof checker **40** – and associated text, e.g., Col. 13: lines 25 – 35; "FIG. 5 is a diagram illustrating an implementation of the proof checker module **40** of FIG. 3. The function of the proof checker module **40** is to verify that the proof **38** supplied by the untrusted proof producer uses only allowed axioms and inference rules and that it is a proof of the required safety predicate **34** ... ". VCGen and proof checker performs the checking of the executable based on the provided specification is performed statically.)

-- Claim 22.

Necula discloses *a computer readable medium storing computer executable components of an executable code check system* (Fig. 6, Col 16: lines 25 – 30; "Software verification modules **66**, of the type disclosed herein in conjunction with the present invention, are stored in the computer storage devices **64**... ".) *comprising*:

- *an input component that receives an object file and a specification associated with the object file*;

(Figs. 3 and 4 – VCGen module **32** – and associated text, e.g., Col. 4: lines 36 – 38; "The VCGen module **32** performs two tasks. First, it checks simple safety properties of the annotated executable **30** ...Second, the VCGen module **32** watches for instructions whose execution might violate the safety policy."

Examiner notes that the object file is synonymous to the executable file, and the specification associated with the object file is the annotation embedded in the executable and/or rules defined by the safety policy.)

• *the specification comprising information associated with a plug-in condition for a method*; and

(Fig. 4 – configuration data **50** – and associated text , e.g., Col. 7: lines 19 – 32; "...The configuration data **50** also describes, by precondition-postcondition pairs, all of the functions that the untrusted code **42** are permitted to invoke.")

• *a checker component that employs the specification to facilitate static checking of the object file, the checker component providing information if a fault condition is determined.*

(Fig. 4 – VCGen 32 – and associated text, e.g., Col. 13: lines 16 – 25; "The safety predicate **34** for a function is obtained by evaluating it symbolically starting in a state that maps the global variables ...and function formal parameters to new variables ... ". Figs. 3 and 5 – proof checker **40** – and associated text, e.g., Col. 13: lines 25 – 35; "FIG. 5 is a diagram illustrating an implementation of the proof checker module **40** of FIG. 3. The function of the proof checker module **40** is to verify that the proof **38** supplied by the untrusted proof producer uses only allowed axioms and inference rules and that it is a proof of the required safety predicate **34** ... ". VCGen and proof checker performs the checking of the executable based on the provided specification is performed statically.)

-- Claim 23.

Necula discloses *an executable code check system* (Fig. 6, computing system) *comprising*:

- *means for receiving a specification associated with an object file, the specification comprising information associated with a plug-in condition for a method*;

(Figs. 3 and 4 – VCGen module **32** – and associated text, e.g., Col. 4: lines 36 – 38; "The VCGen module **32** performs two tasks. First, it checks simple safety properties of the annotated executable **30** ...Second, the VCGen module **32** watches for instructions whose execution might violate the safety policy."

Fig. 4 – configuration data **50** – and associated text , e.g., Col. 7: lines 19 – 32; "...The configuration data **50** also describes, by precondition-postcondition pairs, all of the functions that the untrusted code **42** are permitted to invoke."

Examiner notes that the specification associated with the object file is the annotation embedded in the executable and/or rules defined by the safety policy.)

- *means for statically checking the object file based, at least in part, upon the specification and determining if a fault condition exists*; and

(Fig. 4 – VCGen 32 – and associated text, e.g., Col. 13: lines 16 – 25; "The safety predicate **34** for a function is obtained by evaluating it symbolically starting in a state that maps the global variables ...and function formal parameters to new variables ... ".

Figs. 3 and 5 – proof checker **40** – and associated text, e.g., Col. 13: lines 25 – 35; "FIG. 5 is a diagram illustrating an implementation of the proof checker module **40** of FIG. 3. The function of the proof checker module **40** is to verify that the proof **38** supplied by the untrusted proof producer uses only allowed axioms and inference rules and that it is a proof of the required safety predicate **34** ... ". VCGen and proof checker performs the checking of the executable based on the provided specification is performed statically.)

- *means for providing information if a fault condition is determined to exist.*

(Fig. 4 – Safety Predicate (SP) – and associate text, e.g. Col. 4: lines 40 – 48; "the VCGen module **32** emits a predicate that expresses the conditions under which the execution of the instruction is safe ... ".)

-- Claim 24.

Necula discloses *a method of performing static checking of executable code comprising*:

- *receiving a request, the request including a parameter*,

(Fig. 4 - VCGen **32** – and associate text, e.g. Co. 5: lines 60 – 65; "...Both the precondition and postcondition are parameters of the VCGen module **32** and are part of the safety policy." Thus VCGen receives pre/post condition parameters from configuration data file.)

- *setting a type of a result of a method call to a type of the parameter*, and

(The method or function call is directly controlled by the parameter, i.e. pre/post conditions; thus they must be of the same type.)

- *employing the parameter only during static checking of the method.*

(Fig. 4 – VCGen 32 – and associated text, e.g., Col. 13: lines 16 – 25; "The safety predicate **34** for a function is obtained by evaluating it symbolically starting in a state that maps the global variables ...and function formal parameters to new variables ... ".

Figs. 3 and 5 – proof checker **40** – and associated text, e.g., Col. 13: lines 25 – 35; "FIG. 5 is a diagram illustrating an implementation of the proof checker module **40** of FIG. 3. The function of the proof checker module **40** is to verify that the proof **38** supplied by the untrusted proof producer uses only allowed axioms and inference rules and that it is a proof of the required safety predicate **34** ... ". VCGen and proof checker performs the checking of the executable based on the provided specification is performed statically.)

8. Claims 17 and 18 are rejected under 35 U.S.C. 102(b) as being anticipated by **Tichelaar** (IDS: "A Meta-model for Language-Independent Refactoring").

-- Claim 17.

Tichelaar discloses *a method of developing a software component comprising*:

• *implementing a subclass of a custom state class*;

(Fig. 1: "Class" and "InheritanceDefinition")

• *implementing at least one of a plug-in precondition and a plug-in postcondition as a method of the subclass*;

(Table 1: pre/post conditions in "Add Method".)

• *placing a custom attribute on an enclosing type declaration that references the custom state sub class*; and

(Table 1: "Add Attribute" and "Add Parameter" belonging to a sub class.)

• *placing an attribute on a declaration that references the at least one of a plug-in precondition and a plug-in postcondition.*

(The pre/post conditions provide constraints for a declaration of a class or subclass in Table 1. Thus, "Add Attribute" to the class or subclass with the pre/post conditions as parameters would provide the declaration of that class or subclass with a reference to the pre/post conditions.)

-- Claim 18.

Tichelaar discloses a method of developing a software component of claim 17, however does not explicitly disclose

- A computer readable medium having stored thereon computer executable instructions for carrying out the method of claim 17.

However, it would have been obvious to one of ordinary skill in the art at the time the invention was made to realize that the method of claim 17 needs to be stored on a computer readable medium in order for the functionality it is intended to perform to be realized by a computer.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

9. Claim 7 is rejected under 35 U.S.C. 103(a) as being obvious over **Necula et al**. (US 6,128,774) in view of **Meijer et al** (IDS: "Technical Overview of the Common Language Runtime").

-- Claim 7.

Necula discloses *the system of claim 1,* but does not explicitly disclose

- *the object file being based, at least in part, upon at least one of C#, Visual Basic.net and Managed C++.*

However, **Necula** discloses the executable, i.e. object file, is compiled from typesafe and/or non-typesafe languages (Col. 1: lines 40 – 55; "High level type-safe programming languages, such as ML and Java ... in practice programs often have some components written in ML or Java and other components written in different languages (e.g. C or assembly language.").

Meijer et al. "Technical Overview of the Common Language Runtime", CLR, which is expressed in the Common Intermediate Language (CIL), can be compiled from a language such as C, Pascal, C#, etc. (Introduction).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to use one of C#, Visual Basic.net and Managed C++ as a programming language in **Necula**'s invention because these statically typed OO languages are well supported by Microsoft's Common Language Infrastructure (CLI), as disclosed in **Meijer**, which in turn provides strong support for CLR.

10. Claims 8 – 11 are rejected under 35 U.S.C. 103(a) as being obvious over **Necula et al**. (US 6,128,774) in view of **Das et al.** (IDS: "ESP: Path-Sensitive Program Verification in Polynomial Time").

-- Claim 8.

Necula discloses *the system of claim 1*, however, does not disclose

- *the specification comprising information associated with a state-machine protocol.*

Das discloses a system for qualifying temporal safety property of a program behavior using Finite State Machine (FSM). The system instruments the program under verification by mapping calls to library functions to transition in the property FSM (Fig. 1 and associated text, e.g. page 58).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to associate annotations and/or pre/post conditions of **Necula** with a FSM, as disclosed by **Das**, to thoroughly check for temporal safety properties of the program, for example, verification of all possible execution paths of the program for proper transitional conditions and states.

-- Claim 9.

Necula and **Das** disclose *the system of claim 8*,

- *a state of an object modeled with a custom state*. **Das** further discloses (Page 58, 2nd Col.: Path-sensitive analysis. The states are implemented and instrumented to the program; thus, a state is modeled with a custom state.)

-- Claim 10.

Necula and **Das** disclose the system of claim 9, **Das** further discloses

- the state of the object further being modeled with a custom state component.

(Fig. 3, page 59: Intra-procedural property analysis. The states are software components instrumented in the program.)

-- Claim 11.

Necula and **Das** disclose *the system of claim 10,* **Necula** further discloses *the specification comprising*

- *at least one of a plug-in precondition and a plug-in postcondition method, which is a method of the custom state that is invoked by the checker to perform interface-specific state checks and state transitions.*

(**Necula**: Fig. 4 – configuration data **50** – and associated text , e.g., Col. 7: lines 19 – 32; "...The configuration data **50** also describes, by precondition-postcondition pairs, all of the functions that the untrusted code **42** are permitted to invoke." Since pre/post conditions pair can be function calls, when implementing an FSM with plug-in conditions, the pre/post conditions must also consist of method calls for interface-specific state checks and state transitions as disclosed in **Das** page 63: "The specification includes an FSM that encodes the property to be checked, a set of source code patterns that indicate how fragments of source code map to transitions in the property FSM, and a set of patterns that indicate how fresh stateful values are created by the program".)

11. Claim 14 is rejected under 35 U.S.C. 103(a) as being obvious over **Necula et al**. (US 6,128,774) in view of **Goldberg et al.** (US 6,571,232).

-- Claim 14.

Necula discloses *the system of claim 1*, however, does not disclose the system further comprising

- *a specification extractor that queries a database for its schema and stores information associated with the schema in a specification repository.*

Goldberg discloses a query object generator system comprising

- *a specification extractor that queries a database for its schema and stores information associated with the schema in a specification repository.*

(Fig. 4 and associated text, e.g. Col. 6: lines 40 – 63; "...The query object generator tool **400** includes a mechanism (not shown) for obtaining the database schema from database **404** ... ". Fig. 6 and associated text, e.g. Col. 33: lines 5 – 13; "The query object internal state object **602** allows the user to save a logical definition of a query object in an intermediate file, such as file **612** ... ".)

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to allow the software verification system of **Necula** to query a database for its schema and store information associated with it as disclosed in **Goldberg** so that the system can further verify that command instructions in the program which are associated with database querying are valid.

Conclusion

The prior art made of record and not relied upon is considered pertinent to Applicant's disclosure. See the attached Notice of References Cited.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Thai Van Pham whose telephone number is (571) 270-1064. The examiner can normally be reached on Monday - Thursday, 8am - 3pm EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

TVP

TUAN DAM
SUPERVISORY PATENT EXAMINER